SUBQUERIES

/* SUBQUERIES Learn how subqueries can be used to filter data, generate values, and construct temporary data sets.

A subquery is a query contained within another SQL statement

When the containing statement has finished executing, the data returned by any subqueries is discarded, making a subquery act like a temporary table with statement scope (meaning that the server frees up any memory allocated to the subquery results after the SQL statement has finished execution).*/

```
SELECT
    customer_id, first_name, last_name
FROM
    customer
WHERE
    customer_id = (SELECT
            max(customer_id)
        FROM
            customer);


SELECT
    max(customer_id)
FROM
    customer;


SELECT
    customer_id, first_name, last_name
FROM
    customer
WHERE
    customer_id = 599;
```

/***Subquery Types*** some subqueries are completely self-contained (called noncorrelated subqueries), while others reference columns from the containing statement (called correlated subqueries).*/

/*Non Related Subqueries

The example from earlier is a noncorrelated subquery; it may be executed alone and does not reference anything from the containing statement. Most subqueries that you encounter will be of this type unless you are writing update or delete statements, which frequently make use of correlated subqueries (more on this later). Along with being noncorrelated, the example from earlier in the chapter also returns a result set containing a single row and column. This type of subquery is known as a scalar subquery and can appear on either side of a condition using the usual operators (=, <>, <, >, <=, >=).*/

```
SELECT
    city_id, city
FROM
    city
WHERE
    country_id <> (SELECT
            country_id
        FROM
            country
        WHERE
            country = 'India');


SELECT
    country_id
FROM
    country
WHERE
    country = 'India';
```

```
SELECT
    country_id,city_id, city
FROM
    city
WHERE
    country_id = (SELECT
            country_id
        FROM
            country
        WHERE
            country = 'India');


SELECT
    city_id, city
FROM
    city
WHERE
    country_id <> (SELECT
            country_id
        FROM
            country
        WHERE
            country <> 'India');


/*The containing query fails because an expression (country_id) cannot
be equated to a set of expressions (country_ids 1, 2, 3, ..., 109).
In other words, a single thing cannot be equated to a set of things.*/
```

/*The in and not in operators

While you can’t equate a single value to a set of values, you can check to see whether a single value can be found within a set of values.*/

```
SELECT
    country_id
FROM
    country
WHERE
    country IN ('Canada' , 'Mexico');
```

/*The following query uses the in operator with a subquery on the righthand side of the filter condition to return all cities that are in Canada or

Mexico:*/

```
desc city;

SELECT
    city_id, city
FROM
    city
WHERE
    country_id IN (SELECT
            country_id
        FROM
            country
        WHERE
            country IN ('canada' , 'mexico'));
```

```
SELECT
    city_id, city
FROM
    city
WHERE
    country_id NOT IN (SELECT
            country_id
        FROM
            country
        WHERE
            country IN ('canada' , 'mexico'));
```

/*The all operator

While the in operator is used to see whether an expression can be found within a set of expressions, the all operator allows you to make comparisons between a single value and every value in a set.*/

```
SELECT
    first_name, last_name
FROM
    customer
WHERE
    customer_id <> ALL (SELECT
            customer_id
        FROM
            payment
        WHERE
            amount = 0);
```

/*Without using the all operator*/

```
SELECT
    first_name, last_name
FROM
    customer
WHERE
    customer_id NOT IN (SELECT
            customer_id
        FROM
            payment
        WHERE
            amount = 0);
```

/*All operator with having clause*/

```
SELECT
    customer_id, COUNT(*)
FROM
    rental
GROUP BY customer_id
HAVING COUNT(*) > ALL (SELECT
        COUNT(*)
    FROM
        rental AS r
            INNER JOIN
        customer AS c ON r.customer_id = c.customer_id
            INNER JOIN
        address AS a ON c.address_id = a.address_id
            INNER JOIN
        city AS ct ON a.city_id = ct.city_id
            INNER JOIN
        country AS co ON ct.country_id = co.country_id
    WHERE
        co.country IN ('United STates' , 'Mexico', 'canada')
    GROUP BY r.customer_id);
```

```
SELECT
    c.customer_id,COUNT(*)
FROM
    rental AS r
        INNER JOIN
    customer AS c ON r.customer_id = c.customer_id
        INNER JOIN
    address AS a ON c.address_id = a.address_id
        INNER JOIN
    city AS ct ON a.city_id = ct.city_id
        INNER JOIN
    country AS co ON ct.country_id = co.country_id
WHERE
    co.country IN ('United STates' , 'Mexico', 'canada')
GROUP BY r.customer_id;
```

/*The any operator

Like the all operator, the any operator allows a value to be compared to the members of a set of values; unlike all, however, a condition using the any operator evaluates to true as soon as a single comparison is favorable. This Query will find all customers whose total film rental payments exceed the total payments for all customers in Bolivia, Paraguay, or Chile:*/

```
SELECT
    customer_id, SUM(amount)
FROM
    payment
GROUP BY customer_id
HAVING SUM(amount) > ANY (SELECT
        SUM(p.amount)
    FROM
        payment AS p
            INNER JOIN
        customer AS c ON p.customer_id = c.customer_id
            INNER JOIN
        address AS a ON c.address_id = a.address_id
            INNER JOIN
        city AS ct ON a.city_id = ct.city_id
            INNER JOIN
        country AS co ON ct.country_id = co.country_id
    WHERE
        co.country IN ('Bolivia' , 'Paraguay', 'chile')
    GROUP BY co.country);
```

```
SELECT
    SUM(p.amount)
FROM
    payment AS p
        INNER JOIN
    customer AS c ON p.customer_id = c.customer_id
        INNER JOIN
    address AS a ON c.address_id = a.address_id
        INNER JOIN
    city AS ct ON a.city_id = ct.city_id
        INNER JOIN
    country AS co ON ct.country_id = co.country_id
WHERE
    co.country IN ('Bolivia' , 'Paraguay', 'chile')
GROUP BY co.country;
```

/*Multicolumn Subqueries*/

```
SELECT
    fa.actor_id, fa.film_id
FROM
    film_actor fa
WHERE
    fa.actor_id IN (SELECT
            actor_id
        FROM
            actor
        WHERE
            last_name = 'MONROE')
        AND fa.film_id IN (SELECT
            film_id
        FROM
            film
        WHERE
            rating = 'PG');

SELECT
    actor_id, first_name, last_name
FROM
    actor
WHERE
    last_name = 'monroe';
```

```
/*However, you could merge the two single-column subqueries into one
multicolumn subquery  and compare the results to two columns in the
film_actor table.*/


SELECT
    actor_id, film_id
FROM
    film_actor
WHERE
    (actor_id , film_id) IN (SELECT
            a.actor_id, f.film_id
        FROM
            actor a
                CROSS JOIN
            film f
        WHERE
            a.last_name = 'MONROE'
                AND f.rating = 'PG');
```

/*Correlated Subqueries

All of the subqueries shown thus far have been independent of their containing statements, meaning that you can execute them by themselves and inspect the results. A correlated subquery, on the other hand, is dependent on its containing statement from which it references one or more columns. Unlike a noncorrelated subquery, a correlated subquery is not executed once prior to execution of the containing statement; instead, the correlated subquery is executed once for each candidate row (rows that might be included in the final results).*/

```
SELECT
    c.first_name, c.last_name
FROM
    customer AS c
WHERE
    20 = (SELECT
            COUNT(*)
        FROM
            rental r
        WHERE
            r.customer_id = c.customer_id);


SELECT
    c.first_name, COUNT(*)
FROM
    rental AS r
        INNER JOIN
    customer AS c USING (customer_id)
GROUP BY first_name
HAVING COUNT(*) = 20;
```

```
SELECT
    c.first_name, c.last_name
FROM
    customer AS c
WHERE
    (SELECT
            SUM(p.amount)
        FROM
            payment AS p
        WHERE
            p.customer_id = c.customer_id) BETWEEN 180 AND 240;
```

/*The exists Operator

the most common operator used to build conditions that utilize correlated subqueries is the exists operator. You use the exists operator when you want to identify that a relationship exists without regard for the quantity*/

```
SELECT
    c.first_name, c.last_name
FROM
    customer c
WHERE
    EXISTS( SELECT
            100
        FROM
            rental r
        WHERE
            r.customer_id = c.customer_id
                AND DATE(r.rental_date) < '2005-05-25');
```

/*Using the exists operator, your subquery can return zero, one, or many rows, and the condition simply checks whether the subquery returned one or more rows. If you look at the select clause of the subquery, you will see that it consists of a single literal (1); since the condition in the containing query only needs to know how many rows have been returned, the actual data the subquery returned is irrelevant. Your subquery can return whatever strikes your fancy*/

```
SELECT
    c.first_name, c.last_name
FROM
    customer c
WHERE
    EXISTS( SELECT
            r.rental_date, r.customer_id, 'ABCD' str, 2 * 3 / 7 nmbr
        FROM
            rental r
        WHERE
            r.customer_id = c.customer_id
                AND DATE(r.rental_date) < '2005-05-25');
```

```
/*However, the convention is to specify either select 1 or select *
when using exists.*/


SELECT
    a.first_name, a.last_name
FROM
    actor a
WHERE
    NOT EXISTS( SELECT
            1
        FROM
            film_actor fa
                INNER JOIN
            film f ON f.film_id = fa.film_id
        WHERE
            fa.actor_id = a.actor_id
                AND f.rating = 'R');
```